3. Partial differential equations

In these notes we discuss how to solve numerically partial differential equations (PDEs), with the method of finite difference. Problems with partial differential equations are solved differently according to whether they are *initial value problems*, where the initial conditions (say at $t = 0$) are specified, or *boundary value problem*, where the values on some boundaries are specified. We will consider these two main classes of PDE problem below, focussing on some concrete examples in both cases.

3.1 Initial value problems: discretisation of the diffusion equation

Let us consider the diffusion equation, first in 1 dimension (1D), as our first example of initial value problem. The equation we want to solve is

$$\frac{\partial c(x,t)}{\partial t} = D\frac{\partial^2 c(x,t)}{\partial x^2}, \tag{1}$$

subject to the initial condition $c(x, t = 0) = c_0(x)$. Here and in all the discussion of initial value problems, we imagine that there are periodic boundary conditions in space.

Finite difference methods are based on discretisations of both space and time. In what follows, we will associate j to the spatial discretisation index, and n to the time discretisation – we will also call $c(j; n)$ the discrete approximation of, say, $c(x, t)$. Two successive points in the spatial discretisation of c are separated by δx, while two time steps are separated by δt. In other words we perform the following mapping

$$\begin{aligned} x &\rightarrow j\delta x \\ t &\rightarrow n\delta t. \end{aligned} \tag{2}$$

We now need to specify how to approximate derivatives by finite differences, i.e. how to discretise the PDE in question.

First, the time derivative can be discretised as follows:

$$\frac{\partial c(x,t)}{\partial t} \simeq \frac{c(x,t+\delta t) - c(x,t)}{\delta t} = \frac{c(j;n+1) - c(j;n)}{\delta t}, \tag{3}$$

which is known as forward explicit (also Euler) finite difference rule. The reason for the name "explicit" will become more obvious below.

Now we need to discretise spatial derivatives. To this end, let us perform the following Taylor expansions for $\delta x \rightarrow 0$,

$$\begin{aligned} c(x+\delta x,t) &\simeq c(x) + \delta x\frac{\partial c}{\partial x} + \frac{\delta x^2}{2}\frac{\partial^2 c}{\partial x^2} \\ c(x-\delta x,t) &\simeq c(x) - \delta x\frac{\partial c}{\partial x} + \frac{\delta x^2}{2}\frac{\partial^2 c}{\partial x^2}. \end{aligned} \tag{4}$$

By summing the two expansions in Eq. 4, we obtain

$$c(x+\delta x,t)+c(x-\delta x,t)\simeq 2c(x,t)+\delta x^2\frac{\partial^2 c}{\partial x^2}+\mathcal{O}(\delta x^4) \tag{5}$$

where the error is of order δx^4 because all the odd terms cancel when summing the two expansions in Eq. 4. The discretisation for the (1D) Laplacian is then

$$\frac{\partial^2 c}{\partial x^2}\simeq\frac{c(j+1;n)+c(j-1;n)-2c(j;n)}{\delta x^2}+\mathcal{O}(\delta x^2). \tag{6}$$

This discretisation is an example of a centred difference rule. Another centred difference estimate, this time of $\frac{\partial c}{\partial x}$, is obtained by substracting out the two expansions in Eq. 4,

$$\frac{\partial c}{\partial x}\simeq\frac{c(j+1;n)-c(j-1;n)}{2\delta x}+\mathcal{O}(\delta x^2). \tag{7}$$

Note that the error is again of order δx^2. Note that in Eqs. 6 and 7 we are assuming to be away from the boundaries, while $j+1$ and $j-1$ refer to the right and left boundary respectively.

We can put together the discretisation rules in Eqs. 3 and 6 to obtain the following finite difference approximation for the 1D diffusion equation, Eq. 1,

$$c(j;n+1)=c(j;n)+\frac{D\delta t}{\delta x^2}\left[c(j+1;n)+c(j-1;n)-2c(j;n)\right]. \tag{8}$$

Eq. 8 provides an *explicit* (hence the name of the discretisation in Eq. 3) way to compute the solution at time $n+1$ knowing the solution at time n.

3.2 von Neumann stability analysis

Now we have a finite difference approximation for our PDE and associated initial value problem: it is given by Eq. 8. This is good, but are we sure that the algorithm will be stable, or that it will converge to the exact solution? Intuitively, we expect that if δx and δt are small enough, all should be well. However, it is crucial to be able to answer this question more precisely if we want our PDE algorithm to be of any use in practice!

To understand the stability and convergence properties of our finite difference discretisation, we now perform a calculation, which goes under the name of *von Neumann stability analysis*. Suppose that our numerical solution is equal to the exact solution, $c_{\text{exact}}(x,t)$, plus a numerical error, $\epsilon(x,t)$: $c(x,t)=c_{\text{exact}}(x,t)+\epsilon(x,t)$. As $c_{\text{exact}}(x,t)$ obeys Eq. 1, its discretised version will obey Eq. 8 (this is true in general of any linear PDE). Therefore, as $c(j;n)$ also obeys Eq. 8 (it is the definition of our algorithm!), so will the discretisation of the error term, $\epsilon(j;n)$. Therefore, one has

$$\epsilon(j;n+1)=\epsilon(j;n)+\frac{D\delta t}{\delta x^2}\left[\epsilon(j+1;n)+\epsilon(j-1;n)-2\epsilon(j;n)\right]. \tag{9}$$

Let us now expand $\epsilon(x,t)$ in Fourier series

$$\epsilon(x,t) = \sum_m e^{ik_m x} A_m(t) \tag{10}$$

where k_m is an integer multiple of $2\pi/L$, with L being the size of the integration/simulation domain of integration. As the equation is linear, it suffices to follow the evolution of each of the Fourier component separately. The discretised version of one such component is

$$\epsilon(j;n) = \lambda^n e^{ikj\delta x} \tag{11}$$

where we have assumed $A_m(t) = \lambda^n$: this is because we expect that as $n \to \infty$ the error either increases (if $\lambda > 1$) or decreases to zero (if $\lambda < 1$). This simple form is appropriate in view of the linearity of the PDE in Eq. 1.

Plugging in the ansatz in Eq. 11 into Eq. 9, we obtain the following "dispersion relation" which determines λ as a function of the wavevector k,

$$\lambda = 1 + \frac{D\delta t}{\delta x^2}\left[2\cos(k\delta x) - 2\right]. \tag{12}$$

Numerical stability requires the error not to "explode" with increasing n: this is equivalent to requiring $|\lambda| \leq 1$. As $0 \leq [2 - 2\cos(k\delta x)] \leq 2$, we obtain that λ is always smaller than 1, so the stability criterion is

$$\lambda = 1 - \frac{D\delta t}{\delta x^2}\left[2 - 2\cos(k\delta x)\right] \geq -1. \tag{13}$$

This needs to hold for any k, otherwise some wavelength will grow exponentially and the numerical error will blow up: therefore one needs $1 - 4\frac{D\delta t}{\delta x^2} \geq -1$, or equivalently $\frac{D\delta t}{\delta x^2} \leq \frac{1}{2}$. This classical criterion for the stability of numerical diffusion equations stresses that, if we need to decrease δx to better resolve spatial details in the dynamics, δt needs to be scaled with δx^2 to ensure stability.

3.3 The advection equation

Let us consider as another example the following equation, modelling 1D advection at a constant velocity v,

$$\frac{\partial c}{\partial t} + v\frac{\partial c}{\partial x} = 0. \tag{14}$$

Starting with an initial condition, $c(x, t = 0) = c_0(x)$, Eq. 14 simply advects it with velocity v, $c(x,t) = c_0(x - vt)$.

By combining Eq. 3 and Eq. 7, we can easily write down the following explicit finite difference discretisation for Eq. 14,

$$c(j;n+1) = c(j;n) - \frac{v\delta t}{2\delta x}\left[c(j+1;n) - c(j-1;n)\right]. \tag{15}$$

All seems fine: however a stability analysis as in 3.2 reveals a nasty surprise! Using the ansatz in Eq. 11 one obtains the following dispersion relation linking the growth rate of the error, λ, to the wavevector, k,

$$\lambda = 1 - i\frac{v\delta t}{\delta x}\sin(k\delta x) \tag{16}$$

which leads to a modulus of λ equal to

$$|\lambda| = \sqrt{1 + \left(\frac{v\delta t}{\delta x}\right)^2 \sin^2(k\delta x)} \tag{17}$$

which is always ≥ 1! Therefore the simple and intuitive finite difference discretisation in Eq. 15 is always unstable (in jargon we say that the algorithm defined by Eq. 15 is "unconditionally" unstable).

In order to avoid this instability, the simplest "fix" is to substitute $c(j;n)$ in the right-hand side of Eq. 15 with $\frac{c(j+1;n)+c(j-1;n)}{2}$, as follows

$$c(j;n+1) = \frac{c(j+1;n)+c(j-1;n)}{2} - \frac{v\delta t}{2\delta x}\left[c(j+1;n) - c(j-1;n)\right]. \tag{18}$$

This is known as the Lax method (or prescription). The associated stability criterion can now be found to be $\frac{v\delta t}{\delta x} \leq 1$. The Lax method introduces numerical diffusion, therefore if one starts with a sharp kink at $t = 0$, the interface will widen over time. An even better, but more complicated, choice is to use other discretisation schemes for the advection term $v\frac{\partial c}{\partial x}$, in particular the so-called third-order "upwind" scheme is found to be stable and avoid numerical diffusion to a high accuracy. This should be kept in mind if you are to solve an equation with advection in practice; for our purposes we will not discuss these more complex algorithms in these notes, as our Checkpoint will not involve an advective term.

The advection equation provides a simple instructive example which shows how subtle finite difference methods can be! It is often the case that small changes in the discretisation choice, such as Eq. 15 and 18, lead to a very different quality for the resulting algorithm!

3.4 Initial value problems: the Cahn-Hilliard model and phase separation

In the first part of Checkpoint 3, we will consider a non-trivial partial differential equation: the Cahn-Hilliard equation. Unlike the diffusion (and advection) equation previously considered, the Cahn-Hilliard equation is nonlinear, hence in general it cannot be solved analytically. Like for the diffusion equation, though, we can set up a relatively simple finite difference scheme to solve this equation numerically (as we shall see, solving this equation is an initial value problem, hence we can use the methods employed in Section 3.1).

First, we introduce the Cahn-Hilliard (CH) equation (in generic dimension d, although we will work numerically in $d = 2$ for the first part of the Checkpoint). The CH equation describes phase separation in a physical system such

as a water-and-oil emulsion, disregarding effects arising from hydrodynamics (i.e., the non-zero flow field set up in both the water and oil phase during phase separation). The equation is written in terms of a compositional order parameter, $\phi(\mathbf{x}, t)$, which depends on position $\mathbf{x}$ and time t which is, for instance, positive if locally there is more water than oil, and negative otherwise. The overall integral of this order parameter is conserved, as water cannot turn into oil (or vice versa). Therefore, ϕ obeys a conservation law, which can be written down as follows,

$$\frac{\partial \phi(\mathbf{x}, t)}{\partial t} = -\nabla \cdot \mathbf{J} \tag{19}$$

where $\mathbf{J}$ is a flux.

Note that Eq. 19 is the standard form of evolution equation for conserved quantities: indeed, integrating it over volume, we obtain that

$$\frac{d \int d\mathbf{x} \phi(\mathbf{x}, t)}{dt} = 0, \tag{20}$$

so that $\int d\mathbf{x} \phi(\mathbf{x}, t)$ is constant, under the assumption that there is no flux at the boundary of the system. (Note that to get to Eq. 20 we have used the divergence theorem, a.k.a. the Stokes theorem, to convert an integral over volume into a flux.)

Now, for an ideal gas, the natural order parameter is the local density, $\rho(\mathbf{x}, t)$, which is conserved (like the compositional order parameter $\phi(\mathbf{x}, t)$). The flux in an ideal gas is proportional to the derivative of the density, so that the relevant conservation law is

$$\begin{aligned} \frac{\partial \rho(\mathbf{x}, t)}{\partial t} &= -\nabla \cdot \mathbf{J} \\ \mathbf{J} &= -D \nabla \rho, \end{aligned} \tag{21}$$

so that $\rho(\mathbf{x}, t)$ obeys the diffusion equation (with D the diffusion coefficient).

Instead, for a binary mixture of two fluids (such as water and oil), a (diffusive) flux is not driven by gradients in the density, but by gradients in the *chemical potential* $\mu(\mathbf{x}, t)$ (a quantity which measures how the free energy changes with particle number). Correspondingly, while an ideal gas tends to an equilibrium situation where the density is constant, a mixture tends to an equilibrium where the chemical potential is constant. Therefore the phenomenological form of the flux in the CH equations is

$$\mathbf{J} = -M \nabla \mu, \tag{22}$$

where M is called the "mobility" constant (the equivalent of the diffusion coefficient for a mixture). Inserting this equation for $\mathbf{J}$ in Eq. 19, we obtain the following equation, which is the standard form for the Cahn-Hilliard equation,

$$\frac{\partial \phi(\mathbf{x}, t)}{\partial t} = M \nabla^2 \mu(\mathbf{x}, t), \tag{23}$$

and it turns out that a suitable chemical potential to describe phase separation is given by the following expression,

$$\mu(\mathbf{x},t) = -a\phi(\mathbf{x},t) + b\phi(\mathbf{x},t)^3 - \kappa\nabla^2\phi(\mathbf{x},t). \tag{24}$$

While deriving the expression for $\mu(\mathbf{x},t)$ would take us too far away from the scope of our course, we note here that κ is a constant which is related to the surface tension of an oil-water interface, while a, b are positive constants which ensure that $\phi(\mathbf{x},t) = 0$ (the mixed phase, with oil and water mixed together in equal amounts) is unstable, while $\phi(\mathbf{x},t) = \pm\sqrt{a/b}$ (the demixed phase, with either only/mostly oil or only/mostly water at any point) is stable. You can convince yourself that $\phi(\mathbf{x},t) = 0$ is unstable because the equation for $\phi(\mathbf{x},t) \sim 0$ (where we can neglect the term proportional to $\phi(\mathbf{x},t)^3$) would resemble the diffusion equation, but with a negative diffusion coefficient. From the von Neumann (linear stability) analysis we know this case to be (linearly) unstable, so that $\phi(\mathbf{x},t)$ is pushed away from zero: the nonlinear term $b\phi(\mathbf{x},t)^3$ is then needed to keep $\phi(\mathbf{x},t)$ from blowing up, as would be the case for the simple diffusion equation with negative D, whereas the term proportional to κ avoids oscillations at very small scales ($\sim \delta x$). It is customary to choose $a = b$, which simplifies the algebra as the demixed phase corresponds to $\phi(\mathbf{x},t) = \pm 1$: we will employ this choice in our Checkpoint.

Now, how do we go about discretising Eq. 23? To proceed, we use the same techniques as for the diffusion equation. First, though, it is convenient to discretize the chemical potential. If we work in 2 dimensions, and label position in 2D by indices i and j (corresponding to discretised space along x and y respectively), we can use a centred difference formula for the Laplacian to obtain the estimate

$$\begin{aligned} \mu(i,j;n) &= -a\phi(i,j;n) + b\phi(i,j;n)^3 \\ &- \frac{\kappa}{\delta x^2}\Big[\phi(i+1,j;n) + \phi(i-1,j;n) \\ &+ \phi(i,j+1;n) + \phi(i,j-1;n) - 4\phi(i,j;n)\Big] \end{aligned} \tag{25}$$

where δx is the spatial discretisation (note this is the same along x and y, and as previously we consider a lattice point away from the boundary). As in previous Sections, n labels discretised time. The only difference with respect to the centred difference formula used in 3.1 is that we have discretised the second derivatives over x and y, and summed them up, to obtain the Laplacian.

Having discretised μ, it is now straightforward to see how to discretise Eq. 23, by using centred differences for $\nabla^2\mu$, and forward differences for $\frac{\partial\phi(\mathbf{x},t)}{\partial t}$,

$$\begin{aligned} \phi(i,j;n+1) &= \phi(i,j;n) + \frac{M\delta t}{\delta x^2}\Big[\mu(i+1,j;n) + \mu(i-1,j;n) \\ &+ \mu(i,j+1;n) + \mu(i,j-1;n) - 4\mu(i,j;n)\Big]. \end{aligned} \tag{26}$$

While we do not perform a von Neumann stability analysis in this case, as for the diffusion equation the value of δt needs to be small enough for the algorithm

to converge (a similar treatment works also for this equation, although deriving the stability criterion is more complicated algebraically).

Eq. 26 can now be coded up. If we start with an initial condition where $\phi(\mathbf{x}, t) = 0$ with some small noise, we should find patterns where oil and water demix: i.e., domains with $\phi(\mathbf{x}) \simeq \pm 1$ (if we choose $a = b$) should appear. The route to phase separation involves a dynamical process known as "spinodal decomposition", where oil and water domains can both span the whole simulation domain (this is expressed by saying that the pattern is a "bicontinuous" phase). If, instead, $\phi(\mathbf{x}, t)$ is initialised as non-zero, e.g. $\phi(\mathbf{x}, t) = 0.5$ with noise, bubbles of one phase appear and merge over time. As an exercise, try and think why the patterns are different (to answer, the crucial thing to remember is that the overall integral of $\phi(\mathbf{x}, t)$ is conserved by the dynamics!). The dynamical patterns are quite fascinating to watch, we will aim to simulate these in our Checkpoint.

3.5 Boundary value problems: the Jacobi relaxation algorithm

We now consider the case of boundary value problems, where the solution is not time-dependent, rather it is specified on the boundary of the simulation domain. A prototype of such problems is the Poisson equation of electrostatics,

$$\nabla^2 \phi = -\frac{\rho}{\epsilon} \tag{27}$$

where ϕ is the electrostatic potential, ρ is the charge density, $\epsilon = \epsilon_r \epsilon_0$ where ϵ_0 and ϵ_r are the permittivity of vacuum and relative dielectric permettivity respectively. Boundary conditions are specified at the boundary of the simulation domain (in 3D); they can be Dirichlet (specified ϕ), Neumann (specified normal derivative of ϕ), or mixed boundary conditions.

Let us focus for simplicity first on the 1D case. A class of simple, and often used, methods which to solve boundary value problem is that of relaxation methods. These consist in trying to reformulate the boundary value problem as an appropriate initial value problem, which in steady state provides a solution of the problem of interest. In our case, for instance, we may attempt to solve the following initial value problem,

$$\frac{\partial \phi(x,t)}{\partial t} = \frac{\partial^2 \phi(x,t)}{\partial x^2} + \frac{\rho(x)}{\epsilon}. \tag{28}$$

When the dynamics of Eq. 28 reaches steady state, the associated $\phi(x, t)$ provides a valid solution of the boundary value problem in Eq. 27.

A suitable discretisation of Eq. 28 is

$$\phi(j; n+1) = \phi(j; n) + \delta t \frac{[\phi(j+1; n) + \phi(j-1; n) - 2\phi(j; n)]}{\delta x^2} + \delta t \rho(j), \tag{29}$$

where for simplicity we have just taken $\epsilon = 1$. Note that the explicit update for $\phi(j; n+1)$ is computed for j *inside* the simulation domain; whereas every

time a point on the boundary is needed (either at timestep n or $n+1$), the appropriate boundary condition is used (for instance, in the case of Dirichlet boundary conditions, we do not evolve the values of ϕ on the boundaries but just set it equal to what they should be there).

Now, we should point out that we are not interested in accurately simulating the dynamics related to Eq. 28: it is anyway fictitious! All we care about is the steady state, equilibrated, solution, which provides our numerical estimate for the boundary value problem which we are really interested in. With this in mind, we choose the largest possible value of δt, namely $\delta t = \frac{\delta x^2}{2}$, to accelerate convergence: we then obtain,

$$\phi(j; n+1) = \frac{1}{2}\left[\phi(j+1; n) + \phi(j-1; n) + \delta x^2 \rho(j)\right]. \tag{30}$$

This is the *Jacobi* algorithm. It provides a simple explicit rule to calculate $\phi(j; n+1)$ in terms of value of ϕ at the previous timestep n. This algorithm can also be straighforwardly generalised to more dimensions. For example in 3D the Jacobi algorithm is (a position in 3D is labelled by three indices, i, j and k),

$$\begin{aligned} \phi(i,j,k;n+1) &= \frac{1}{6}\Big[\phi(i+1,j,k;n) + \phi(i-1,j,k;n) \\ &+ \phi(i,j+1,k;n) + \phi(i,j-1,k;n) \\ &+ \phi(i,j,k+1;n) + \phi(i,j,k-1;n) + \delta x^2 \rho(i,j,k)\Big]. \end{aligned} \tag{31}$$

The Jacobi algorithm is nice and simple; it can be coded up easily and it is therefore quite popular. Its problem is that its convergence is quite slow: one can prove that in order to reduce the numerical error in the steady state estimate for ϕ by a factor of 10^{-p}, one needs an extra pN^2 iterations, where N is the linear dimension of the discretisation lattice (which in D dimensions would thus consist of N^D points). We will now see how to (slightly) modify the Jacobi algorithm to achieve faster convergence to the steady state solution.

3.6 Boundary value problems: the Gauss-Seidel and successive over-relaxation methods

One simple modification of the Jacobi algorithm comes from the realisation that when n is large it would be thoretically equivalent in Eqs. 30, 31 if, instead of the value at timestep n, we used the value of timestep $n+1$ for ϕ on the right hand side. This leads to the Gauss-Seidel algorithm, which generalises Eq. 30 to

$$\phi(j; n+1) = \frac{1}{2}\left[\phi(j+1; n) + \phi(j-1; n+1) + \delta x^2 \rho(j)\right]. \tag{32}$$

where you should note the $\phi(j-1; n+1)$ on the right hand side. This algorithm assumes that we are looping over j from 0 to $N-1$ when updating ϕ. If this is done, than when updating site j at timestep $n+1$ we know $j+1$ only at timestep n, but we know $j-1$ at both timestep n and $n+1$ (as we have just

updated $j-1$!) Therefore one may imagine that a faster convergence can be attained if we plug in the *latest* estimate for $\phi(j-1)$ on the right hand side. This is indeed correct (although not so easy to prove!), and the Gauss-Seidel algorithm converges twice as fast as the Jacobi algorithm. The Gauss-Seidel algorithm also allows us to only keep track of one array for ϕ, with only its latest value, as the updated values of $\phi(j)$ are used immediately when updating $\phi(j+1)$.

A problem of the Gauss-Seidel algorithm, which is relevant to high performance computing, is that it is not easy to parallelise, as in common parallel architectures we are not sure when a site is updated with respect to another one. The workaround is to divide the lattice in alternating black and white nodes (like a checkerboard in 2D): then one can update first all black, then all whites etc, without worrying about conflicting/simultaneous updates.

While the Gauss-Seidel algorithm is faster, this is still not enough for applications to real-world problem, for instance to situations where charges can move, so that the electrostatic potential needs to be recomputed very often. A trick to achieve even faster convergence is to use the *successive over-relaxation (sor) method*. This is based on the observation that the typical reason why relaxation methods are slow is that the field/variable under investigation changes less and less with n.

Let us first consider a simple relaxation methods, where the target variable, x, is a scalar rather than a field (our electrostatic potential). A relaxation method then provides a recursion relation to go from $x(n)$, the $n-$th estimate of x, to $x(n+1)$, say given by $x(n+1) = f(x(n))$ (f is a generic function). Let us now introduce a "relaxation" parameter ω, and define a new recursion as follows

$$x(n+1) = \omega f(x(n)) + (1-\omega)x(n) = f(x(n)) + (\omega - 1)\delta x(n) \tag{33}$$

where $\delta x(n) = f(x(n)) - x(n)$. The case $\omega = 0$ corresponds to no relaxation; the case $\omega = 1$ to the $x(n+1) = f(x(n))$ recursion; while $0 < \omega < 1$ and $1 < \omega < 2$ correspond respectively to "under-relaxing" and "over-relaxing", as $x(n+1)$ undershoots and overshoots $f(x(n))$ respectively. Typically there is an optimal, and model-dependent, ω value between 1 and 2 which renders convergence as fast as possible within this family of models.

In the boundary value problem we are considering, the analog of $f(x(n))$ is the Gauss-Seidel estimate. The SOR model then is equivalent to the following update rule (in 1D)

$$\phi(j; n+1) = (1-\omega)\phi(j; n) + \omega\phi_{GS}(j; n+1) \tag{34}$$

where $\phi_{GS}(j; n+1)$ denotes the Gauss-Seidel update rule, Eq. 32. The SOR model can be generalised straightforwardly in 3D. You should play around with ω in your code to see what gives the fastest convergence in your case!